

---

# **Trixy Documentation**

***Release 3.0.0b0***

**Austin Hartzheim**

January 04, 2016



<b>1</b>	<b>Getting Started</b>	<b>1</b>
1.1	Installing Trixy . . . . .	1
1.2	The Basics: Theory . . . . .	1
1.3	Modifying Data: Custom Processors . . . . .	2
<b>2</b>	<b>Examples</b>	<b>3</b>
2.1	Passthrough Proxy . . . . .	3
2.2	Changing Website Responses . . . . .	3
2.3	More Eamples Soon . . . . .	4
<b>3</b>	<b>Trixy Code Documentation</b>	<b>5</b>
3.1	trixy . . . . .	5
3.2	trixy.encryption . . . . .	10
3.3	trixy.proxy . . . . .	10
<b>4</b>	<b>What is Trixy?</b>	<b>13</b>
<b>5</b>	<b>Other Documentation</b>	<b>15</b>
<b>6</b>	<b>Indices and tables</b>	<b>17</b>
	<b>Python Module Index</b>	<b>19</b>



---

## Getting Started

---

Here are some instructions for getting started with Trixy.

### 1.1 Installing Trixy

Installing Trixy is just a simple command. Note that you should use the Python 3 version:

```
sudo pip install trixy
```

Alternatively, you can download a source tarball or zip file from [PyPI](#) or [Github](#). Then, you can extract it and install it by running:

```
sudo python3 setup.py install
```

### 1.2 The Basics: Theory

Trixy is structured into four component classes: servers, inputs, outputs, and processors. Servers are responsible for capturing incoming connections and passing them to an input class. The input class then takes these connections and builds processing chains for them. These processing chains consist of processors, which modify data passing through them, and outputs, which forward the data stream (including any modifications) to a remote host.

To use Trixy, you should import it into your Python project and create subclasses of `trixy.TrixyInput`. Inside the `__init__()` method of the subclass, you should create a chain of nodes which the data should pass through. As an example:

```
def __init__(self, loop):
    super().__init__(loop)

    processor = trixy.TrixyProcessor()
    self.connect_node(processor)
    processor.connect_node(trixy.TrixyOutput(loop, '127.0.0.1', 9999))
```

The first line creates a processor node. The default `trixy.TrixyProcessor` class does not do anything other than forward the data, so you should create a subclass and override some of its methods to modify its behavior (covered next). The second line connects the input instance with this processor node so that the input will forward the data it gets to the processor. The last line connects the processor node to a `trixy.TrixyOutput` instance that is created at the same time. This causes the processor to forward data it gets to the output (after making any modifications). The default output that is used in this case creates a TCP connection to localhost on port 9999 and forwards the data there.

## 1.3 Modifying Data: Custom Processors

Trixy is great for simply re-routing data, but its real power lies in its ability to process the data on the fly. To do this, you need to create a custom `trixy.TrixyProcessor` subclass.

When you are creating your own custom processor, you should modify packets like so:

```
class CustomProcessor(trixy.TrixyProcessor):
    def handle_packet_down(self, data):
        # Modify the data variable here
        self.forward_packet_down(data)

    def handle_packet_up(self, data):
        # Modify the data variable here
        self.forward_packet_up(data)
```

The `handle_packet_down()` method is called to process data flowing from the output to the input. The `handle_packet_up()` method is used to process data moving from the input to the output. The calls to the `forward_packet_down()` and `forward_packet_up()` then send the modified data on its way to the next node(s) in the chain.

---

**Note:** It is also the case that you can omit calls to `forward_packet_down()` and `forward_packet_up()` when you want to drop a packet.

---

---

## Examples

---

Here are some examples of how to use Trixy:

### 2.1 Passthrough Proxy

The following code creates a Trixy proxy server on local port 8080 and then sends the output to austinhardtzhaim.me on port 80:

```
# /usr/bin/env python3
import asyncio
import trixy

class CustomInput(trixy.TrixyInput):
    def __init__(self, loop):
        super().__init__(loop)

        # This output class automatically connects to austinhardtzhaim.me
        output = trixy.TrixyOutput(loop, 'austinhardtzhaim.me', 80)
        self.connect_node(output)

if __name__ == '__main__':
    # Run the Trixy server on localhost, port 8080
    loop = asyncio.get_event_loop()
    coro = loop.create_server(lambda: CustomInput(loop),
                              '127.0.0.1', 8080)

    loop.create_task(coro)
    loop.run_forever()
```

This example was modified from the [README file](#).

### 2.2 Changing Website Responses

The following example takes an incoming connection on a localhost port 8080, redirects it to example.com on port 80, and then modifies the response from example.com:

```
#!/usr/bin/env python3
import asyncio
import trixy

REMOTE_ADDR = '93.184.216.34' # IP for example.com
```

```
REMOTE_PORT = 80

class ExampleReplacer(trixy.TrixyProcessor):

    def handle_packet_up(self, data):
        '''
        The HTTP headers will show that we are connecting to 127.0.0.1
        on port 8080, which will cause example.com to return an error.
        So, we fix this to show the expected header.
        '''
        data = data.replace(b'127.0.0.1:8080', b'example.com')
        self.forward_packet_up(data)

    def handle_packet_down(self, data):
        '''
        Check all incoming packets; replace some basic information on
        the page with our own information. For example, replace the
        "Example Domain" heading with the text "Trixy Example Domain".
        '''
        data = data.replace(b'Example Domain', b'Trixy Example Domain')
        data = data.replace(b'://www.iana.org/domains/example',
                           b's://github.com/austinhartzheim/Trixy')
        self.forward_packet_down(data)

class CustomInput(trixy.TrixyInput):

    def __init__(self, loop):
        super().__init__(loop)

        processor = ExampleReplacer()
        self.connect_node(processor)

        output = trixy.TrixyOutput(loop, REMOTE_ADDR, REMOTE_PORT)
        processor.connect_node(output)

if __name__ == '__main__':
    # Bind to localhost, port 8080
    loop = asyncio.get_event_loop()
    coro = loop.create_server(lambda: CustomInput(loop),
                              '127.0.0.1', 8080)
    loop.create_task(coro)
    loop.run_forever()
```

This example was originally posted [on the developer's website](#).

## 2.3 More Eamples Soon

More examples are on their way! But, if you write one first, feel free to send a pull request on [Github](#).



---

## Trixy Code Documentation

---

### 3.1 trixy

The main Trixy module contains the parent classes that can be modified for custom functionality.

**class** `trixy.TrixyInput` (*loop*)

Bases: `trixy.TrixyNode`, `asyncio.protocols.Protocol`

Once a connection is open, establish an output chain.

**add\_downstream\_node** (*node*)

Add a one direction downstream link to the node parameter.

**Parameters** *node* (`TrixyNode`) – The downstream node to create a unidirectional link to.

**add\_upstream\_node** (*node*)

Add a one direction upstream link to the node parameter.

**Parameters** *node* (`TrixyNode`) – The upstream node to create a unidirectional link to.

**connect\_node** (*node*)

Create a bidirectional connection between the two nodes with the downstream node being the parameter.

**Parameters** *node* (`TrixyNode`) – The downstream node to create a bidirectional connection to.

**connection\_lost** (*ex*)

The connection from the client was lost. Close the chain.

**connection\_made** (*transport*)

An incoming connection has been established; save the transport.

**data\_received** (*data*)

Received data from the client; forward it up the chain.

**eof\_received** ()

Called when the other end calls `write_eof()` or equivalent.

If this returns a false value (including `None`), the transport will close itself. If it returns a true value, closing the transport is up to the protocol.

**forward\_packet\_down** (*data*)

Forward data to all downstream nodes.

**Parameters** *data* (*bytes*) – The data to forward.

**forward\_packet\_up** (*data*)

Forward data to all upstream nodes.

**Parameters** **data** (*bytes*) – The data to forward.

**handle\_packet\_down** (*data*)

Take a downward-bound packet and forward it to the client.

**handle\_packet\_up** (*data*)

Handle data moving upwards. TrixyProcessor children should perform some action on *data* whereas *Trixy-Output* children should send the data to the desired output location.

Generally, the a child implementation of this method should be implemented such that it calls `self.forward_packet_down` with the data (post-modification if necessary) to forward the data to other processors in the chain. However, if the processor is a filter, it may drop the packet by omitting that call.

**Parameters** **data** (*bytes*) – The data that is being handled.

**pause\_writing** ()

Called when the transport's buffer goes over the high-water mark.

Pause and resume calls are paired – `pause_writing()` is called once when the buffer goes strictly over the high-water mark (even if subsequent writes increases the buffer size even more), and eventually `resume_writing()` is called once when the buffer size reaches the low-water mark.

Note that if the buffer size equals the high-water mark, `pause_writing()` is not called – it must go strictly over. Conversely, `resume_writing()` is called when the buffer size is equal or lower than the low-water mark. These end conditions are important to ensure that things go as expected when either mark is zero.

NOTE: This is the only Protocol callback that is not called through `EventLoop.call_soon()` – if it were, it would have no effect when it's most needed (when the app keeps writing without yielding until `pause_writing()` is called).

**resume\_writing** ()

Called when the transport's buffer drains below the low-water mark.

See `pause_writing()` for details.

**class** `trixy.TrixyNode`

Bases: `object`

A base class for TrixyNodes that implements some default packet forwarding and node linking.

**add\_downstream\_node** (*node*)

Add a one direction downstream link to the node parameter.

**Parameters** **node** (`TrixyNode`) – The downstream node to create a unidirectional link to.

**add\_upstream\_node** (*node*)

Add a one direction upstream link to the node parameter.

**Parameters** **node** (`TrixyNode`) – The upstream node to create a unidirectional link to.

**connect\_node** (*node*)

Create a bidirectional connection between the two nodes with the downstream node being the parameter.

**Parameters** **node** (`TrixyNode`) – The downstream node to create a bidirectional connection to.

**forward\_packet\_down** (*data*)

Forward data to all downstream nodes.

**Parameters** **data** (*bytes*) – The data to forward.

**forward\_packet\_up** (*data*)

Forward data to all upstream nodes.

**Parameters** **data** (*bytes*) – The data to forward.

**handle\_close** (*direction*='down')

The connection has closed on one end. So, shutdown what we are doing and notify the nodes we are connected to.

**Parameters** *direction* (*str*) – ‘down’ or ‘up’ depending on if downstream nodes need to be closed, or upstream nodes need to be closed.

**handle\_packet\_down** (*data*)

Handle data moving downwards. TrixyProcessor children should perform some action on *data* whereas TrixyOutput children should send the data to the desired output location.

Generally, the a child implementation of this method should be implemented such that it calls self.forward\_packet\_down with the data (post-modification if necessary) to forward the data to other processors in the chain. However, if the processor is a filter, it may drop the packet by omitting that call.

**Parameters** *data* (*bytes*) – The data that is being handled.

**handle\_packet\_up** (*data*)

Handle data moving upwards. TrixyProcessor children should perform some action on *data* whereas TrixyOutput children should send the data to the desired output location.

Generally, the a child implementation of this method should be implemented such that it calls self.forward\_packet\_down with the data (post-modification if necessary) to forward the data to other processors in the chain. However, if the processor is a filter, it may drop the packet by omitting that call.

**Parameters** *data* (*bytes*) – The data that is being handled.

**class** trixy.TrixyOutput (*loop*, *host*, *port*, *autoconnect*=True)

Bases: trixy.TrixyNode, asyncio.protocols.Protocol

Output node; generally to connect to another network service.

**add\_downstream\_node** (*node*)

Add a one direction downstream link to the node parameter.

**Parameters** *node* (TrixyNode) – The downstream node to create a unidirectional link to.

**add\_upstream\_node** (*node*)

Add a one direction upstream link to the node parameter.

**Parameters** *node* (TrixyNode) – The upstream node to create a unidirectional link to.

**connect** ()

Connect to the saved host and port. This method is commonly called by \_\_init\_\_() when autoconnect is enabled.

**connect\_node** (*node*)

Create a bidirectional connection between the two nodes with the downstream node being the parameter.

**Parameters** *node* (TrixyNode) – The downstream node to create a bidirectional connection to.

**connection\_lost** (*exc*)

Called when the connection is lost or closed.

The argument is an exception object or None (the latter meaning a regular EOF is received or the connection was aborted or closed).

**connection\_made** (*transport*)

Outbound connection successful; save the transport.

**data\_received** (*data*)

Received incoming data; forward it down the chain.

**Parameters** *data* (*bytes*) – a bytes object of the received data.

**eof\_received()**

Called when the other end calls `write_eof()` or equivalent.

If this returns a false value (including `None`), the transport will close itself. If it returns a true value, closing the transport is up to the protocol.

**forward\_packet\_down(data)**

Forward data to all downstream nodes.

**Parameters** **data** (*bytes*) – The data to forward.

**forward\_packet\_up(data)**

If the transport is connected, send the data onwards; otherwise it will be added to the buffer.

This method require buffering because data often becomes available immediately for upload, but the transport needs time to connect. (This is especially the case when `autoconnect` is set to `True`).

**handle\_close(direction='down')**

The connection has closed on one end. So, shutdown what we are doing and notify the nodes we are connected to.

**Parameters** **direction** (*str*) – ‘down’ or ‘up’ depending on if downstream nodes need to be closed, or upstream nodes need to be closed.

**handle\_packet\_down(data)**

Hadle data moving downwards. `TrixyProcessor` children should perform some action on *data* whereas *TrixyOutput* children should send the data to the desired output location.

Generally, the a child implementation of this method should be implemented such that it calls `self.forward_packet_down` with the data (post-modification if necessary) to forward the data to other processors in the chain. However, if the processor is a filter, it may drop the packet by omitting that call.

**Parameters** **data** (*bytes*) – The data that is being handled.

**handle\_packet\_up(data)**

Hadle data moving upwards. `TrixyProcessor` children should perform some action on *data* whereas *TrixyOutput* children should send the data to the desired output location.

Generally, the a child implementation of this method should be implemented such that it calls `self.forward_packet_down` with the data (post-modification if necessary) to forward the data to other processors in the chain. However, if the processor is a filter, it may drop the packet by omitting that call.

**Parameters** **data** (*bytes*) – The data that is being handled.

**pause\_writing()**

Called when the transport’s buffer goes over the high-water mark.

Pause and resume calls are paired – `pause_writing()` is called once when the buffer goes strictly over the high-water mark (even if subsequent writes increases the buffer size even more), and eventually `resume_writing()` is called once when the buffer size reaches the low-water mark.

Note that if the buffer size equals the high-water mark, `pause_writing()` is not called – it must go strictly over. Conversely, `resume_writing()` is called when the buffer size is equal or lower than the low-water mark. These end conditions are important to ensure that things go as expected when either mark is zero.

NOTE: This is the only Protocol callback that is not called through `EventLoop.call_soon()` – if it were, it would have no effect when it’s most needed (when the app keeps writing without yielding until `pause_writing()` is called).

**resume\_writing()**

Called when the transport’s buffer drains below the low-water mark.

See `pause_writing()` for details.

```
class trixy.TrixyProcessor
```

Bases: `trixy.TrixyNode`

Perform processing on data moving through Trixy.

```
add_downstream_node (node)
```

Add a one direction downstream link to the node parameter.

**Parameters** `node` (`TrixyNode`) – The downstream node to create a unidirectional link to.

```
add_upstream_node (node)
```

Add a one direction upstream link to the node parameter.

**Parameters** `node` (`TrixyNode`) – The upstream node to create a unidirectional link to.

```
connect_node (node)
```

Create a bidirectional connection between the two nodes with the downstream node being the parameter.

**Parameters** `node` (`TrixyNode`) – The downstream node to create a bidirectional connection to.

```
forward_packet_down (data)
```

Forward data to all downstream nodes.

**Parameters** `data` (`bytes`) – The data to forward.

```
forward_packet_up (data)
```

Forward data to all upstream nodes.

**Parameters** `data` (`bytes`) – The data to forward.

```
handle_close (direction='down')
```

The connection has closed on one end. So, shutdown what we are doing and notify the nodes we are connected to.

**Parameters** `direction` (`str`) – ‘down’ or ‘up’ depending on if downstream nodes need to be closed, or upstream nodes need to be closed.

```
handle_packet_down (data)
```

Handle data moving downwards. `TrixyProcessor` children should perform some action on `data` whereas `TrixyOutput` children should send the data to the desired output location.

Generally, the a child implementation of this method should be implemented such that it calls `self.forward_packet_down` with the data (post-modification if necessary) to forward the data to other processors in the chain. However, if the processor is a filter, it may drop the packet by omitting that call.

**Parameters** `data` (`bytes`) – The data that is being handled.

```
handle_packet_up (data)
```

Handle data moving upwards. `TrixyProcessor` children should perform some action on `data` whereas `TrixyOutput` children should send the data to the desired output location.

Generally, the a child implementation of this method should be implemented such that it calls `self.forward_packet_down` with the data (post-modification if necessary) to forward the data to other processors in the chain. However, if the processor is a filter, it may drop the packet by omitting that call.

**Parameters** `data` (`bytes`) – The data that is being handled.

```
class trixy.TrixyServer (tinput, host, port, loop=None)
```

Bases: `object`

Main server to grab incoming connections and forward them.

```
close ()
```

Shutdown the server.

**run\_loop()**

Run the event loop so that the server functions.

## 3.2 trixy.encryption

The Trixy encryption module holds inputs and outputs that have support for encryption that applications might expect. For example, the `trixy.encryption.TrixySSLInput` can be used to trick a browser into thinking it is creating an encrypted connection, but the connection can then be re-routed through an unencrypted `trixy.TrixyOutput` for easier monitoring.

**class trixy.encryption.TrixySSLInput** (*sock, addr, \*\*kwargs*)

Acts like a normal TrixyInput, but uses Python's `ssl.wrap_socket()` code to speak the SSL protocol back to applications that expect it.

**class trixy.encryption.TrixySSLOutput** (*loop, host, port, autoconnect=True, \*\*kwargs*)

Acts like a normal TrixyOutput, but uses Python's `ssl.wrap_socket()` code to speak the SSL protocol to servers that expect it.

By default this class allows for SSL2 and SSL3 connections in addition to TLS. If you want to specify different settings, you can pass your own context to `setup_socket()`.

**class trixy.encryption.TrixyTLSOutput** (*loop, host, port, autoconnect=True, \*\*kwargs*)

Acts identical to a TrixySSLOutput, but defaults to only accepting TLS for security reasons. This makes it slightly easier to prevent downgrade attacks, especially when doing hasty testing rather than full development.

## 3.3 trixy.proxy

The Trixy proxy inputs speak a variety of common proxy protocols, such as SOCKS4, SOCKS4a, and SOCKS5. Their default behavior is to act as a normal proxy and open a connection to the desired endpoint. However, this behavior can be overridden to create different results.

Additionally, the proxy outputs allow a connection to be subsequently made to a proxy server. This allows intercepted traffic to be easily routed on networks that require a proxy. It also makes it easier to route traffic into the Tor network.

**class trixy.proxy.Socks4Input** (*sock, addr*)

Implements the SOCKS4 protocol as defined in this document: <http://www.openssh.com/txt/socks4.protocol>

**handle\_connect\_request** (*addr, port, userid*)

The application connecting to this SOCKS4 input has requested that a connection be made to a remote host. At this point, that request can be accepted, modified, or declined.

The default behavior is to accept the request as-is.

**handle\_proxy\_request** (*data*)

In SOCKS4, the first packet in a connection is a request to either initiate a connection to a remote host and port, or it is a request to bind a port. This method is responsible for processing those requests.

**reply\_request\_failed** (*addr, port*)

Send a reply stating that the request was rejected (perhaps due to a firewall rule forbidding the connection or binding) or that it failed (i.e., the remote host could not be connected to or the requested port could not be bound).

**reply\_request\_granted** (*addr, port*)

Send a reply stating that the connection or bind request has been granted and that the connection or bind attempt was successfully completed.

**reply\_request\_rejected** (*addr, port*)

Send a reply saying that the request was rejected because the SOCKS server could not connect to the client's identd server.

**reply\_request\_rejected\_id\_mismatch** (*addr, port*)

Send a reply saying that the request was rejected because the SOCKS server was sent an ID by the client that did not match the ID returned by identd on the client's computer.

**class** trixy.proxy.**Socks4aInput** (*sock, addr*)

Implements the SOCKS4a protocol, which is the same as the SOCKS4 protocol except for the addition of DNS resolution as described here: <http://www.openssh.com/txt/socks4a.protocol>

**handle\_connect\_request** (*addr, port, userid*)

The application connecting to this SOCKS4 input has requested that a connection be made to a remote host. At this point, that request can be accepted, modified, or declined.

The default behavior is to accept the request as-is.

**handle\_proxy\_request** (*data*)

In SOCKS4, the first packet in a connection is a request to either initiate a connection to a remote host and port, or it is a request to bind a port. This method is responsible for processing those requests.

**class** trixy.proxy.**Socks5Input** (*loop*)

Implements the SOCKS5 protocol as defined in RFC1928. At present, only CONNECT requests are supported.

**handle\_connect\_request** (*addr, port, addrtype*)

The application connecting to this SOCKS4 input has requested that a connection be made to a remote host. At this point, that request can be accepted, modified, or declined.

The default behavior is to accept the request as-is.

**handle\_method\_select** (*methods*)

Select the preferred authentication method from the list of client-supplied supported methods. The byte object of length one should be sent to self.reply\_method to notify the client of the method selection.

**reply\_method** (*method*)

Send a reply to the user letting them know which authentication method the server has selected. If the method 0xff is selected, close the connection because no method is supported.

**reply\_request\_granted** (*addr, port, addrtype*)

Send a reply stating that the connection or bind request has been granted and that the connection or bind attempt was successfully completed.

**class** trixy.proxy.**Socks5Output** (*host, port, autoconnect=True, proxyhost='127.0.0.1', proxy-port=1080*)

Implements the SOCKS5 protocol as defined in RFC1928.

**handle\_state\_change** (*oldstate, newstate*)

Be able to process events when they occur. It allows easier detection of when events occur if it is desired to implement different responses. It also allows detection of when the proxy is ready for use and can be used to use assume\_connected to transfer control to a TrixyOutput.

#### Parameters

- **oldstate** (*int*) – The old state number.
- **newstate** (*int*) – The new state number.

**exception** trixy.proxy.**SocksProtocolError**

Someone sent some invalid data on the wire, and this is how to deal with it.





---

## **What is Trixy?**

---

Trixy is designed to be used in a variety of situations involving network traffic interception, injection, and modification. The software allows you to easily get your code running between two endpoints of a network connection. This allows you to easily:

- Log protocols for reverse engineering.
- Modify packets on bidirectional connections.
- Inject traffic into a network connection.
- Develop and test protocol parsers.
- Monitor applications for suspicious network activity.
- Sanitize traffic, removing any undesired information.
- Develop application level firewalls.

Here are some practical examples of the above:

- Cheating at video games:
  - Exploit server-client trust by modifying packets indicating how much money a player has.
  - Drop packets that indicate damage to a player.
- Removing advertising and trackers from webpages.
- Performing man-in-the-middle attacks.



---

## Other Documentation

---

If you are stuck, you should also check the following sources for information about Trixy:

- [The developer's website](#)
- [The Github repository](#)



---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`



## t

`trixy`, [5](#)

`trixy.encryption`, [10](#)

`trixy.proxy`, [10](#)





## A

`add_downstream_node()` (trixy.TrixyInput method), 5  
`add_downstream_node()` (trixy.TrixyNode method), 6  
`add_downstream_node()` (trixy.TrixyOutput method), 7  
`add_downstream_node()` (trixy.TrixyProcessor method), 9  
`add_upstream_node()` (trixy.TrixyInput method), 5  
`add_upstream_node()` (trixy.TrixyNode method), 6  
`add_upstream_node()` (trixy.TrixyOutput method), 7  
`add_upstream_node()` (trixy.TrixyProcessor method), 9

## C

`close()` (trixy.TrixyServer method), 9  
`connect()` (trixy.TrixyOutput method), 7  
`connect_node()` (trixy.TrixyInput method), 5  
`connect_node()` (trixy.TrixyNode method), 6  
`connect_node()` (trixy.TrixyOutput method), 7  
`connect_node()` (trixy.TrixyProcessor method), 9  
`connection_lost()` (trixy.TrixyInput method), 5  
`connection_lost()` (trixy.TrixyOutput method), 7  
`connection_made()` (trixy.TrixyInput method), 5  
`connection_made()` (trixy.TrixyOutput method), 7

## D

`data_received()` (trixy.TrixyInput method), 5  
`data_received()` (trixy.TrixyOutput method), 7

## E

`eof_received()` (trixy.TrixyInput method), 5  
`eof_received()` (trixy.TrixyOutput method), 7

## F

`forward_packet_down()` (trixy.TrixyInput method), 5  
`forward_packet_down()` (trixy.TrixyNode method), 6  
`forward_packet_down()` (trixy.TrixyOutput method), 8  
`forward_packet_down()` (trixy.TrixyProcessor method), 9  
`forward_packet_up()` (trixy.TrixyInput method), 5  
`forward_packet_up()` (trixy.TrixyNode method), 6  
`forward_packet_up()` (trixy.TrixyOutput method), 8  
`forward_packet_up()` (trixy.TrixyProcessor method), 9

## H

`handle_close()` (trixy.TrixyNode method), 6  
`handle_close()` (trixy.TrixyOutput method), 8  
`handle_close()` (trixy.TrixyProcessor method), 9  
`handle_connect_request()` (trixy.proxy.Socks4aInput method), 11  
`handle_connect_request()` (trixy.proxy.Socks4Input method), 10  
`handle_connect_request()` (trixy.proxy.Socks5Input method), 11  
`handle_method_select()` (trixy.proxy.Socks5Input method), 11  
`handle_packet_down()` (trixy.TrixyInput method), 6  
`handle_packet_down()` (trixy.TrixyNode method), 7  
`handle_packet_down()` (trixy.TrixyOutput method), 8  
`handle_packet_down()` (trixy.TrixyProcessor method), 9  
`handle_packet_up()` (trixy.TrixyInput method), 6  
`handle_packet_up()` (trixy.TrixyNode method), 7  
`handle_packet_up()` (trixy.TrixyOutput method), 8  
`handle_packet_up()` (trixy.TrixyProcessor method), 9  
`handle_proxy_request()` (trixy.proxy.Socks4aInput method), 11  
`handle_proxy_request()` (trixy.proxy.Socks4Input method), 10  
`handle_state_change()` (trixy.proxy.Socks5Output method), 11

## P

`pause_writing()` (trixy.TrixyInput method), 6  
`pause_writing()` (trixy.TrixyOutput method), 8

## R

`reply_method()` (trixy.proxy.Socks5Input method), 11  
`reply_request_failed()` (trixy.proxy.Socks4Input method), 10  
`reply_request_granted()` (trixy.proxy.Socks4Input method), 10  
`reply_request_granted()` (trixy.proxy.Socks5Input method), 11

`reply_request_rejected()` (trixy.proxy.Socks4Input method), 10  
`reply_request_rejected_id_mismatch()` (trixy.proxy.Socks4Input method), 11  
`resume_writing()` (trixy.TrixyInput method), 6  
`resume_writing()` (trixy.TrixyOutput method), 8  
`run_loop()` (trixy.TrixyServer method), 9

## S

`Socks4aInput` (class in trixy.proxy), 11  
`Socks4Input` (class in trixy.proxy), 10  
`Socks5Input` (class in trixy.proxy), 11  
`Socks5Output` (class in trixy.proxy), 11  
`SocksProtocolError`, 11

## T

`trixy` (module), 5  
`trixy.encryption` (module), 10  
`trixy.proxy` (module), 10  
`TrixyInput` (class in trixy), 5  
`TrixyNode` (class in trixy), 6  
`TrixyOutput` (class in trixy), 7  
`TrixyProcessor` (class in trixy), 8  
`TrixyServer` (class in trixy), 9  
`TrixySSLInput` (class in trixy.encryption), 10  
`TrixySSLOutput` (class in trixy.encryption), 10  
`TrixyTLSOutput` (class in trixy.encryption), 10